

# Efficient Information Flow Maximization in Probabilistic Graphs

Christian Frey<sup>1</sup>, Andreas Züfle<sup>2</sup>, Tobias Emrich<sup>1</sup>, Matthias Renz<sup>3</sup>

<sup>1</sup>Institute for Informatics, Ludwig-Maximilians-Universität München

<sup>2</sup>Dept. of Geography and Geoinformation Science, George Mason University

<sup>3</sup>Dept. of Computational and Data Sciences, George Mason University

frey@dbs.ifi.lmu.de, azufle@gmu.edu, emrich@dbs.ifi.lmu.de, mrenz@gmu.edu

## ABSTRACT

Reliable propagation of information through large networks, e.g. communication networks, social networks or sensor networks is very important in many applications concerning marketing, social networks, and wireless sensor networks. However, social ties of friendship may be obsolete, and communication links may fail, inducing the notion of uncertainty in such networks. In this paper, we address the problem of optimizing information propagation in uncertain networks given a constrained budget of edges. We show that this problem requires to solve two NP-hard subproblems: the computation of expected information flow, and the optimal choice of edges. To compute the expected information flow to a source vertex, we propose the *F-tree* as a specialized data structure, that identifies independent components of the graph for which the information flow can either be computed analytically and efficiently, or for which traditional Monte-Carlo sampling can be applied independent of the remaining network. For the problem of finding the optimal edges, we propose a series of heuristics that exploit properties of this data structure. Our evaluation shows that these heuristics lead to high quality solutions, thus yielding high information flow, while maintaining low run-time.

## 1. INTRODUCTION

Nowadays, social and communication networks have become ubiquitous in our daily life to receive and share information. Whenever we are navigating the World Wide Web, updating our social network profiles, or sending a text message on our cell-phone, we participate in an information network as a node. In such networks, network nodes exchange some sort of information: In social networks, users share their opinions and ideas, aiming to convince others. In wireless sensor networks nodes collect data and aim to ensure that this data is propagated through the network: Either to a destination, such as a server node, or simply to as many other nodes as possible. Abstractly speaking, in all of these networks, nodes aim at propagating their information, or their belief, throughout the network. The event of a successful propagation of information between nodes is subject to inherent uncertainty. In a wireless sensor, telecommunication or electrical network, a link can be unreliable and may fail with certain probability [10, 32]. In a social network, trust and influence issues may impact the likelihood of social interactions or the likelihood of convincing another of an individual's

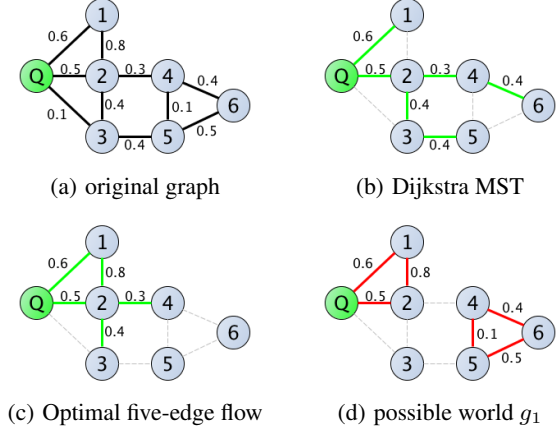


Figure 1: Running example.

idea [11, 18, 1]. The probabilistic graph model is commonly used to address such scenarios in a unified way (e.g. [25, 28, 15, 29, 41, 40]). In this model, each edge is associated with an existential probability to quantify the likelihood that this edge exists in the graph. Traditionally, to maximize the likelihood of a successful communication between two nodes, information is propagated by flooding it through the network. Thus, every node that receives a bit of information will proceed to share this information with all its neighbors. Clearly, such a flooding approach is not applicable for large communication networks and for social networks, as the communication between two network nodes incurs a cost: Sensor network nodes, e.g. in micro-sensor networks, have limited computing capability, memory resources and power supply, require battery power to send, receive and forward messages, and are also limited by their bandwidth; individuals of a social network require time and sometimes even additional monetary resources to convince others of their ideas.

In this work, we address the following problem: Given a probabilistic network graph  $\mathcal{G}$  with edges that can be activated for communication, i.e. enabled to transfer information, or stay inactive. The problem is to send/receive information from a single node  $Q$  in  $\mathcal{G}$  to/from as many nodes in  $\mathcal{G}$  as possible assuming a limited budget of edges that can be activated. To solve this problem, the main focus is on the selection of edges to be activated.

**Example 1.** To illustrate our problem setting, consider the network depicted in Figure 1(a). The task is to maximize the information flow to node  $Q$  from other nodes given a limited budget of edges to be used. In contrast to the general problem defined later, this example assumes equal weights of all nodes. Each edge of the network is labeled with a probability value denoting the probability of a successful communication. A straightforward solution to this problem, is to activate all edges. Assuming each node to have one unit of

information, the expected information flow of this solution can be shown to be  $\simeq 2.51$ . While maximizing the information flow, this solution incurs the maximum possible communication cost. A traditional trade-off between these single-objective solutions is using a probability maximizing Dijkstra's spanning tree, as depicted in Figure 1(b). The expected information flow in this setting can be shown to aggregate to 1.59 units, while requiring six edges to be activated. Yet, it can be shown that the solution depicted in Figure 1(c) dominates this solution: Only five edges are used, thus further reducing the communication cost, while achieving a higher expected information flow of  $\simeq 2.02$  units of information to  $Q$ .

The aim of this work is to efficiently find a near-optimal sub-network, which maximizes the expected flow of information at a constrained budget of edges. In Example 1, we computed the information flow for various example graphs. But in fact, this computation has been shown to be exponentially hard in the number of edges of the graph, and thus impractical to be solved analytically. Furthermore, the optimal selection of edges to maximize the information flow is shown to be NP-hard. These two subproblems define the main computational challenges addressed in this work.

To tackle these challenges, the remainder of this work is organized as follows. After a survey of related work in Section 2, we recapitulate common definitions for stochastic networks and formally define our problem setting in Section 3. After a more detailed technical overview in Section 4, the theoretical heart of this work is presented in Section 5. We show how to identify independent subgraphs, for which the information flow can be computed independently. This allows to divide the main problem into much smaller sub-problems. To conquer these subproblems, we identify cases for which the expected information flow can be computed analytically, and we propose to employ Monte-Carlo sampling to approximate the information flow of the remaining cases. Section 5.3 is the algorithmic core of our work, showing how aforementioned independent components can be organized hierarchical in a *F-tree* which is inspired by the *block-cut tree* [36, 14, 38]. This structure allows us to aggregate results of individual components efficiently, and we show how previous Monte-Carlo sampling results can be re-used as more edges are selected and activated. Our experimental evaluation in Section 7 shows that our algorithms significantly outperform traditional solutions, in terms of combined communication cost and information flow, on synthetic and real stochastic networks. In summary, the main contributions of this work are:

- Theoretical complexity study of the flow maximization problem in probabilistic graphs.
- Efficient estimation of the expected information flow based on network graph decomposition and Monte-Carlo sampling.
- Our *F-tree* structure enabling efficient organization of independent graph components and (local) intermediate results for efficient expected flow computation.
- An algorithm for iterative selection of edges to be activated to maximize the expected information flow.
- Thorough experimental evaluation of the proposed methods and algorithms.

## 2. RELATED WORK

Mining probabilistic graphs (a.k.a. uncertain graphs) has recently attracted much attention in the data mining and database research communities [19, 39, 20, 29]. We summarize state-of-the-art publications and relate our work in this context.

**Subgraph Reliability.** A related and fundamental problem in uncertain graph mining is the so-called subgraph reliability problem, which asks to estimate the probability that two given (sets

of) nodes are reachable. This problem, well studied in the context of communication networks, has seen a recent revival in the database community due to the need for scalable solutions for big networks. Specific problem formulations in this class ask to measure the probability that two specific nodes are connected (two-terminal reliability [2]), all nodes in the network are pairwise connected (all-terminal reliability [34]), or all nodes in a given subset are pairwise connected (k-terminal reliability [13, 12]). Extending these reliability queries, where source and sink node(s) are specified, the corresponding graph mining problem is to find, for a given probabilistic graph, the set of most reliable k-terminal subgraphs [16]. All these problem definitions have in common that the set of nodes to be reached is predefined, and that there is no degree of freedom in the number of activated edges - thus all nodes are assumed to attempt to communicate to all their neighbors, which we argue can be overly expensive in many applications.

**Reliability Bounds.** Several lower bounds on (two-terminal) reliability have been defined in the context of communication networks [3, 4, 9, 30]. Such bounds could be used in the place of our sampling approach, to estimate the information gain obtained by adding a network edge to the current active set. However, for all these bounds, the computational complexity to obtain these bounds is at least quadratic in the number of network nodes, making these bounds unfeasible for large networks. Very simple but efficient bounds have been presented in [19], such as using the most-probable path between two nodes as a lower bound of their two-terminal reliability. However, the number of possible (non-circular) paths is exponentially large in the number of edges of a graph, such that, in practice, even the most probable path will have a negligible probability, thus yielding a useless upper bound. Thus, since none of these probability bounds are sufficiently effective and efficient for practical use, we directly decided to use a sampling approach for parts of the graph where no exact inference is possible.

**Influential Nodes.** Existing work motivated by applications to marketing provide methods to detect *influential* members within a social network. This can help to promote a new product. The task is to detect nodes, i.e. persons, where the chance that the product is recommended to a broad range of connected people is maximized. In [6], [31] a framework is provided which considers the interactions between the persons in a probabilistic model. As the problem of finding the most influential vertices is NP-hard, approximation algorithms are used in [18], outperforming basic heuristics based on degree centrality and distance centrality which are applied traditionally in social networks. This branch of research has in common that the task is to activate a constrained number of nodes to maximize the information flow, whereas our problem definition constrains the number of activated edges for a single specified query/sink node.

**Reliable Paths.** In mobile ad-hoc networks, the uncertainty of an edge can be interpreted as the connectivity between two nodes. Thus, an important problem in this field is to maximize the probability that two nodes are connected for a constrained budget of edges [10]. In this work, the main difference to our work is that the information flow to a single destination is maximized, rather than the information flow in general. The heuristics [10] cannot be applied directly to our problem, since clearly, maximizing the flow to one node may detriment the flow to another node.

**Bi-connected components.** The *F-tree* that we propose in this work is inspired by the *block-cut tree* [36, 14, 38]. The main difference is that our approach aims at finding cyclic subgraphs, where nodes are bi-connected. For subgraphs having a size of at least three vertices, this problem is equivalent to finding bi-connected subgraphs, which is solved in [36, 14, 38]. Thus, our proposed data

structure treats bi-connected subgraphs of size less than three separately, grouping them together as mono-connected components. More importantly, this existing work does not show how to compute, estimate and propagate probabilistic information through the structure, which is the main contribution of this work.

### 3. PROBLEM DEFINITION

A probabilistic undirected graph is given by  $\mathcal{G} = (V, E, W, P)$ , where  $V$  is a set of vertices,  $E \subseteq V \times V$  is a set of edges,  $W : V \mapsto \mathbb{R}^+$  is a function that maps each vertex to a positive value representing the information weight of the corresponding vertex and  $P : E \mapsto (0, 1]$  is a function that maps each edge to its corresponding probability of existing in  $\mathcal{G}$ . In the following, it is assumed that the existence of different edges are independent from one another. Let us note, that our approach also applies to other models such as the conditional probability model [29], as long as a computational method for an unbiased drawing of samples of the probabilistic graph is available.

In a probabilistic graph  $\mathcal{G}$ , the existence of each edge is a random variable. Thus, the topology of  $\mathcal{G}$  is a random variable, too. The sample space of this random variable is the set of all possible graphs. A possible graph  $g = (V_g, E_g)$  of a probabilistic graph  $\mathcal{G}$  is a deterministic graph which is a possible outcome of the random variables representing the edges of  $\mathcal{G}$ . The graph  $g$  contains a subset of edges of  $\mathcal{G}$ , i.e.,  $E_g \subseteq E$ . The total number of such possible graphs is  $2^{|E|}$ , where  $|E|$  represents the number of edges  $e \in E$  having  $P(e) < 1$ , because for each such edge, we have two cases as to whether or not that edge is present in the graph. We let  $\mathcal{W}$  denote the set of all possible graphs. The probability of sampling the graph  $g$  from the random variables representing the probabilistic graph  $\mathcal{G}$  is given by the following sampling or realization probability  $Pr(g)$ :

$$Pr(g) = \prod_{e \in E_g} P(e) \cdot \prod_{e \in E \setminus E_g} (1 - P(e)). \quad (1)$$

Figure 1(a) shows an example of a probabilistic graph  $\mathcal{G}$  and its possible realization  $g_1$  in 1(d). This probabilistic graph has  $2^{10} = 1024$  possible worlds. Using Equation 1, the probability of world  $g_1$  is given by:

$$Pr(g_1) = 0.6 \cdot 0.5 \cdot 0.8 \cdot 0.4 \cdot 0.4 \cdot 0.5 \cdot (1 - 0.1) \cdot (1 - 0.3) \cdot (1 - 0.4) \cdot (1 - 0.1) = 0.00653184$$

**Definition 1 (Path).** Let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph and let  $v_a, v_b \in V$  be two nodes such that  $v_a \neq v_b$ . An (acyclic) path  $path(v_a, v_b) = v_a, v_1, v_2, \dots, v_b$  is a sequence of vertices, such that  $\forall v_i \in path(v_a, v_b) : (v_i \in V)$  and  $\forall v_i, v_j \in path(v_a, v_b) : v_i \neq v_j$ .

**Definition 2 (Reachability).** The network reachability problem as defined in [15, 5] computes the likelihood of the binomial random variable  $\uparrow(i, j, \mathcal{G})$  of two nodes  $i, j \in V$  being connected in  $\mathcal{G}$ , formally:

$$P(\uparrow(i, j, \mathcal{G})) := \sum_{g \in \mathcal{W}} \prod_{e \in E_g} P(e) \cdot \prod_{e \in E \setminus E_g} (1 - P(e)) \cdot \uparrow(i, j, g),$$

where  $\uparrow(i, j, g)$  is an indicator function that returns one if there exists a path between nodes  $i$  and  $j$  in the (deterministic) possible graph  $g$ , and zero otherwise. For a given query node  $Q$ , our aim is to optimize the information gain, which is defined as the total weight of nodes reachable from  $Q$ .

**Definition 3 (Expected Information Flow).** Let  $Q \in V$  be a node and let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph, then  $flow(Q, \mathcal{G})$  denotes the random variable of the sum of vertex weights of all nodes in  $V$  reachable from  $Q$ , formally:

$$flow(Q, \mathcal{G}) := \sum_{v \in V} P(\uparrow(Q, v, \mathcal{G})) \cdot W(v).$$

Due to linearity of expectations, and exploiting that  $W(v)$  is deterministic, we can compute the expectation  $E(flow(Q, \mathcal{G}))$  of this random variable as  $E(flow(Q, \mathcal{G})) =$

$$E\left(\sum_{v \in V} P(\uparrow(Q, v, \mathcal{G})) \cdot W(v)\right) = \sum_{v \in V} E(P(\uparrow(Q, v, \mathcal{G}))) \cdot W(v) \quad (2)$$

Given our definition of Expected Information Flow in Equation 2, we can now state our formal problem definition of optimizing the expected information flow of a probabilistic graph  $\mathcal{G}$  for a constrained budget of edges.

**Definition 4 (Maximum Expected Information Flow).** Let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph, let  $Q \in V$  be a query node and let  $k$  be a non-negative integer. The Maximum Expected Information Flow

$$MaxFlow(\mathcal{G}, Q, k) =$$

$$argmax_{G=(V, E' \subseteq E, W, P), |E'| \leq k} E(flow(Q, G)), \quad (3)$$

is the subgraph of  $\mathcal{G}$  maximizing the information flow  $Q$  constrained to having at most  $k$  edges.

Computing  $MaxFlow(\mathcal{G}, Q, k)$  efficiently requires to overcome two NP-hard subproblems. First, the computation of the expected information flow  $E(flow(Q, \mathcal{G}))$  to vertex  $Q$  for a given probabilistic graph  $\mathcal{G}$  is NP-hard as shown in [5]. In addition, the problem of selecting the optimal set of  $k$  vertices to maximize the information flow  $MaxFlow(\mathcal{G}, Q, k)$  is a NP-hard problem in itself, as shown in the following.

**Theorem 1.** Even if the Expected Information Flow  $flow(Q, \mathcal{G})$  to a vertex  $Q$  can be computed in  $O(1)$  for any probabilistic graph  $\mathcal{G}$ , the problem of finding  $MaxFlow(\mathcal{G}, Q, k)$  is still NP-hard.

*Proof.* A formal proof can be found in the appendix in Section 10.  $\square$

### 4. ROADMAP

To compute  $MaxFlow(\mathcal{G}, Q, k)$ , we first need an efficient solution to approximate the reachability probability  $E(\uparrow(Q, v, \mathcal{G}))$  from  $Q$  to/from a single node  $v$ . Since this problem can be shown to be #P-hard, Section 5.3 presents an approximation technique which exploits stochastic independencies between branches of a spanning tree of subgraph  $G$  rooted at  $Q$ . This technique allows to aggregate independent subgraphs of  $G$  efficiently, while exploiting a sampling solution for components of the graph  $MaxFlow(\mathcal{G}, Q, k)$  that contains cycles.

Once we can efficiently approximate the flow  $E(\uparrow(Q, v, \mathcal{G}))$  from  $Q$  to each node  $v \in V$ , we next tackle the problem of efficiently finding a subgraph  $MaxEFlow(\mathcal{G}, Q, k)$  that yields a near-optimal expected information flow given a budget of  $k$  edges in Section 6. Due to the theoretic result of Theorem 1, we propose heuristics to choose  $k$  edges from  $\mathcal{G}$ . Finally, our experiments in Section 7 support our theoretical intuition that our solutions for the two aforementioned subproblems synergize: An optimal subgraph will choose a budget of  $k$  edges in a tree-like fashion, to reach large parts of the probabilistic graph. At the same time, our solutions exploit tree-like subgraphs for efficient probability computation.

## 5. EXPECTED FLOW ESTIMATION

In this section we estimate the expected information flow of a given subgraph  $G \subseteq \mathcal{G}$ . Following Equation 2, the reachability probability  $P(\uparrow(Q, v, \mathcal{G}))$  between  $Q$  and a node  $v$  can be used to compute the total expected information flow  $E(\text{flow}(Q, \mathcal{G}))$ . This problem of computing the reachability probability between two nodes has been shown to be  $\#P$ -hard [10, 5] and sampling solutions have been proposed to approximate it [22, 7]. In this section, we will present our solution to identify subgraphs of  $\mathcal{G}$  for which we can compute the information analytically and efficiently, such that expensive numeric sampling only has to be applied to small subgraphs. We first introduce the concept of Monte-Carlo sampling of a subgraph.

### 5.1 Traditional Monte-Carlo Sampling

**Lemma 1.** *Let  $\mathcal{G} = (V, E, W, P)$ , be an uncertain graph and let  $\mathcal{S}$  be a set of sample worlds drawn randomly and unbiased from the set  $\mathcal{W}$  of possible graphs of  $\mathcal{G}$ . Then the average information flow in samples in  $\mathcal{S}$*

$$\frac{1}{|\mathcal{S}|} \sum_{g \in \mathcal{S}} \text{flow}(Q, g) = \frac{1}{|\mathcal{S}|} \cdot \sum_{g \in \mathcal{S}} \sum_v \uparrow(Q, v, g) \cdot W(v) \quad (4)$$

*is an unbiased estimator of the expected information flow  $E(\text{flow}(Q, \mathcal{G}))$ , where  $\uparrow(Q, v, g)$  is an indicator function that returns one if there exists a path between nodes  $Q$  and  $v$  in the (deterministic) sample graph  $g$ , and zero otherwise.*

*Proof.* For  $\mu$  to be an unbiased estimator of  $E(\text{flow}(Q, \mathcal{G}))$ , we have to show that  $E(\mu) = E(\text{flow}(Q, \mathcal{G}))$ . Substituting  $\mu$  yields  $E(\mu) = E(\frac{1}{|\mathcal{S}|} \sum_{g \in \mathcal{S}} \text{flow}(Q, g))$ . Due to linearity of expectations, this is equal to  $\frac{1}{|\mathcal{S}|} \sum_{g \in \mathcal{S}} E(\text{flow}(Q, g))$ . The sum over  $|\mathcal{S}|$  identical values can be replaced by a factor of  $|\mathcal{S}|$ . Reducing this factor yields  $E(\text{flow}(Q, g \in \mathcal{S}))$ . Following the assumption of unbiased sampling  $\mathcal{S}$  from the set  $\mathcal{W}$  of possible worlds, the expected information flow  $E(\text{flow}(Q, g))$  of a sample possible world  $g \in \mathcal{S}$  is equal to the expected information flow  $E(\text{flow}(Q, \mathcal{G}))$ .  $\square$

Naive sampling of the whole graph  $\mathcal{G}$  has disadvantages: First, this approach requires to compute reachability queries on a set of possibly large sampled graphs. Second, a rather large approximation error is incurred. We will approach these drawbacks by first describing how non-cyclic subgraphs, i.e. trees, can be processed in order to compute the information flow exactly and efficiently without sampling. For cyclic subgraphs we show how sampled information flows can be used to compute the information flow in the full graph.

### 5.2 Mono-Connected vs. Bi-Connected graphs

The main observation that will be exploited in our work is the following: if there exists only one possible path between two vertices, then we can compute their reachability probability efficiently.

**Definition 5 (Mono-Connected Nodes).** *Let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph and let  $A, B \in V$ . If  $\text{path}(A, B) = (A = v_1, v_2, \dots, v_{k-1}, v_k = B)$  is the only path between  $A$  and  $B$ , i.e., there exists no other path  $p \in V \times V \times V^*$  that satisfies Definition 1, then we denote  $A$  and  $B$  as mono-connected.*

In the following, when the query vertex  $Q$  is clear from the context, we call a vertex  $A$  mono-connected if it is mono-connected to the query vertex  $Q$ .

**Lemma 2.** *If two vertices  $A$  and  $B$  are mono-connected in a probabilistic graph  $\mathcal{G}$ , then the reachability probability between  $A$  and  $B$  is equal to the product of the edge probabilities included in  $\text{path}(A, B)$ , i.e.,*

$$\uparrow(A, B, \mathcal{G}) = \prod_{i=1}^{k-1} P((v_i, v_{i+1})) \text{ with } v_i \in \text{path}(A, B)$$

*Proof.* Following possible world semantics as defined in Definition 2, the reachability probability  $\uparrow(A, B, \mathcal{G})$  is the sum of probabilities of all possible worlds where  $B$  is connected to  $A$ . We show that  $A$  and  $B$  are connected in a possible graph  $g$  if and only if all  $k-1$  edges  $e_i = (v_i, v_{i+1}) \in \text{path}(A, B)$  exist.

$\Rightarrow$ : By contradiction: Let  $A$  and  $B$  be connected in  $g$ , and let any edge on  $\text{path}(A, B)$  be missing. Then there must exist a path  $\text{path}^{\text{prime}}(A, B) \neq \text{path}(A, B)$  which contradicts the assumption that  $A$  and  $B$  are mono-connected.

$\Leftarrow$ : If all edges on  $\text{path}(A, B)$  exist, then  $B$  is connected to  $A$  following the assumption that  $\text{path}(A, B)$  is a path from  $A$  to  $B$ .

Due to our assumption of independent edges, the probability that all edges in  $\text{path}(A, B)$  exist is given by  $\prod_{i=1}^{k-1} P((v_i, v_{i+1}))$ .  $\square$

**Definition 6 (Mono-Connected Graph).** *A probabilistic graph  $\mathcal{G} = (V, E, W, P)$  is called mono-connected, iff all pairs of vertices in  $V$  are mono-connected.*

Next, we generalize Lemma 2 to whole subgraphs, such that a specified vertex  $Q$  in that subgraph has a unique path to all other vertices in the subgraph. Using Lemma 2, we constitute the following theorem that will be exploited in the remainder of this work.

**Theorem 2.** *Let  $\mathcal{G} = (V, E, \mathcal{G}, P)$  be a probabilistic graph, let  $Q \in V$  be a node. If  $\mathcal{G}$  is mono-connected, then  $E(\text{flow}(Q, \mathcal{G}))$  can be computed efficiently.*

*Proof.*  $E(\text{flow}(Q, \mathcal{G}))$  is the sum of reachability probabilities of all nodes, according to Equation 2. If  $\mathcal{G}$  is connected and non-cyclic, we can guarantee that each node has exactly one path to  $Q$ , and thus, is mono-connected. Thus, Lemma 2 is applicable to compute the reachability probability between  $Q$  and each node  $v \in V$ . Due to linearity of expectations, i.e.,  $E(X + Y) = E(X) + E(Y)$  for random variables  $X$  and  $Y$ , we can aggregate individual reachability expectations, yielding  $E(\text{flow}(Q, \mathcal{G}))$ .  $\square$

Analogously to Definition 5, we define bi-connected nodes.

**Definition 7 (Bi-Connected Nodes).** *Let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph and let  $A, B \in V$ . If there exists (at least) two paths  $\text{path}_1(A, B)$  and  $\text{path}_2(A, B)$  such that  $\text{path}_1(A, B) \neq \text{path}_2(A, B)$ , then we denote  $A$  and  $B$  as bi-connected.*

**Definition 8 (Bi-Connected Graph).** *A bi-connected graph [36, 14] is a connected probabilistic graph  $\mathcal{G} = (V, E, W, P)$  such that removal of any one vertex  $A \in V$  will still yield a connected probabilistic graph.*

**Lemma 3.** *In a bi-connected graph  $\mathcal{G}$  of size  $|V| \geq 3$ , all pairs of vertices are bi-connected following Definition 7.*

*Proof.* A formal proof of Lemma 3 can be found in the appendix in Section 10.  $\square$

The information flow within a bi-connected graph can not be computed efficiently using Theorem 2, as the flow between any two nodes  $A$  to  $B$  is shared by more than one path. In the next section, we propose techniques to substitute bi-connected subgraphs

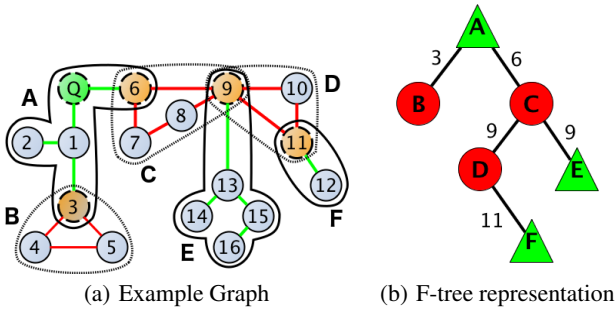


Figure 2: Running example graph with corresponding *F-tree*

by super-nodes, for which we can estimate the information flow using Monte-Carlo sampling exploiting Lemma 1. By substituting the bi-connected subgraphs by super-nodes for which we apply sampling and memorize the sampling information for these super-nodes, we yield a mono-connected graph that uses the substituted super-nodes. This approach maximizes the partitions of the graph for which expensive Monte-Carlo estimation can be replaced using Theorem 2.

The next section will show how to achieve this goal, by employing a *F-tree* of the graph. This data structure borrowed from graph theory partitions the graph into bi-connected components (a.k.a. “blocks”) generated by bi-connected subgraphs, and identifies vertices of the graph as *articulation vertices* to connect two bi-connected components. We exploit these articulation vertices, by having them represent all the information flow that is estimated to flow to them from their corresponding bi-connected component. This approach is described in detail in the following subsection.

### 5.3 Flow tree

In this section we propose to adapt the block-cut tree [36, 14, 38] to partition a graph into independent bi-connected components. Instead of sampling the whole uncertain graph, the purpose of this index structure is to exploit Theorem 2 for mono-connected components, and to apply local Monte-Carlo within bi-connected components only. Our employed *Flow tree* (*F-tree*) memorizes the information flow at each node. Before we show how to utilize the *F-tree* for efficient information flow computation, we first give a formal definition.

**Definition 9** (Flow tree). Let  $\mathcal{G} = (V, E, W, P)$  be a probabilistic graph and let  $Q \in V$  be a vertex for which the expected information flow is to be computed. A Flow tree (*F-tree*) is a tree structure defined as follows.

- 1) each component of the *F-tree* is a connected subgraph of  $\mathcal{G}$ . A component can be mono-connected or bi-connected.
- 2) a mono-connected component  $MC = (MC.V \subseteq V, MC.AV \in V)$  is a set of vertices  $MC.V \cup MC.AV$  that form a mono-connected subgraph (c.f. Definition 6) in  $\mathcal{G}$ . The vertex  $MC.AV$  is called articulation vertex. Intuitively, a mono-connected component represents a tree-like structure rooted at  $MC.AV$ . Using Theorem 2, we can efficiently compute the information flow from all vertices  $MC.V$  to  $MC.AV$ .
- 3) a bi-connected component  $BC = (BC.V, BC.P(v), BC.AV)$  is a set of vertices  $BC.V \cup BC.AV$  of size greater than two that form a bi-connected subgraph  $\mathcal{G}'$  in  $\mathcal{G}$  according to Definition 8. Intuitively, a bi-connected component represents a subgraph describing a cycle. In this case, we can estimate likelihood of being connected to the articulation vertex  $BC.AV$  using Monte-Carlo

sampling in Lemma 1. The function  $BC.P(v) : BC.V \mapsto [0, 1]$  maps each vertex  $v \in BC.V$  to the estimated reachability probability  $\text{reach}(v, BC.AV)$  of  $v$  being connected to  $BC.AV$  in  $\mathcal{G}$ .

4) For each pair of (mono- or bi-connected) components  $(C_1, C_2)$ , it holds that the intersection  $C_1.V \cap C_2.V = \emptyset$  of vertices is empty. Thus, each vertex in  $V$  is mapped to at most one component’s vertex set.

5) Two different components may share the same articulation vertex, and the articulation vertex of one component may be in the vertex set of another component.

6) The articulation vertex of the root of a *F-tree* is  $Q \in V$ .

Intuitively speaking, a component is a set of vertices together with an articulation vertex that all information must flow through in order to reach  $Q$ . By our iterative construction algorithm presented in Section 5.4, each component is guaranteed to have such an articulation vertex, guiding the direction to vertex  $Q$ . The idea of the *F-tree* is to use components as virtual nodes, such that all actual vertices of a component send their information to their articulation vertex. Then the articulation vertex forwards all information to the next component, until the root of the tree is reached where all information is sent to articulation vertex  $Q$ .

**Example 2.** As an example for a *F-tree*, consider Figure 2(a), showing a probabilistic graph. For brevity, assume that each edge  $e \in E$  has an existential probability of  $p(e) = 0.5$  and that all vertices  $v \in V$  have a information weight corresponding to their id, e.g. vertex 6 has a weight of six. A corresponding *F-tree* is shown in Figure 2(b). A mono-connected component is given by  $A = (\{1, 2, 3, 6\}, Q)$ . For this component, we can exploit Theorem 2 to analytically compute the flow of information from any vertex in  $\{1, 2, 3, 6\}$  to articulation vertex  $Q$ : vertices 3 and 6 are connected to  $Q$  with probability 0.5. Thus, these nodes contributed an expected information flow of  $3 \cdot 0.5 = 1.5$  and  $6 \cdot 0.5 = 3$  respectively. Vertices 2 and 3 are connected to  $Q$  with a probability of  $0.5 \cdot 0.5 = 0.25$ , respectively, following Lemma 2. Thus, these nodes contribute an expected information of  $2 \cdot 0.25 = 0.5$  and  $3 \cdot 0.25 = 0.75$ . Following Theorem 2, we can aggregate these probabilities to obtain the expected information flow from vertices  $\{1, 2, 3, 6\}$  to articulation vertex  $Q$  as 5.75.

A bi-connected component is defined by  $B = (\{4, 5\}, 3)$ , representing a sub-graph having a cycle. Having a cycle, we cannot exploit Theorem 2 to compute the flow of a vertex in  $\{4, 5\}$  to vertex 3. But we can sample the subgraph spanned by vertices in  $\{3, 4, 5\}$  to estimate probabilities that vertices  $\{4, 5\}$  are connected to articulation vertex 3 using Lemma 1. With sufficient samples, this will yield a probability of around 0.375 for both vertices. Again using Theorem 2, we compute an information flow of  $0.375 \cdot 4 + 0.375 \cdot 5 = 3.375$  to articulation vertex 3. Given this expected flow, we can use the mono-connected component  $A$  to compute the expected information analytically that is further propagated from the articulation vertex 3 of component  $B$  to the articulation vertex  $Q$  of  $A$ . Since the articulation vertex of component  $B$  is in the vertex set of component  $A$ , component  $B$  is a child of component  $A$  in Figure 2(b) since  $B$  propagates its information to  $A$ . As we have already computed above, the probability of vertex 3 to be connected to its articulation vertex  $Q$  is 0.25, yielding an information flow worth  $3.375 \cdot 0.25 = 0.84375$  units flowing from vertices  $\{4, 5\}$  to  $Q$ . Again, exploiting Theorem 2, we can aggregate this to a total flow of  $5.75 + 0.84375 = 6.59375$  from vertices  $\{1, 2, 3, 4, 5, 6\}$  to  $Q$ .

Another bi-connected component is  $C = (\{7, 8, 9\}, 6)$ , for which we can estimate the information flow from vertices 7, 8, and 9 to articulation vertex 6 numerically using Monte-Carlo sampling. Since

vertex 6 is in  $A$ , component  $C$  is a child of  $A$ . We find another bi-connected component  $D = (\{10, 11\}, 9)$ , and two more mono-connected components  $E = (\{13, \dots, 16\}, 9)$  and  $F = (\{12\}, 11)$ .

In this example, the structure of the  $F$ -tree allows us to compute or approximate the expected information flow to  $Q$  from each vertex. For this purpose, only three small components  $B$  and  $C$  and  $D$  need to be sampled. This is a vast reduction of sampling space compared to a naive Monte-Carlo approach that samples the full graph: rather than sampling a single random variable having  $2^{|E|} = 2^{19} = 524288$  possible worlds, we only need to sample three random variables corresponding to the bi-connected components  $B$ ,  $C$  and  $D$  having  $2^3 = 8$ ,  $2^4 = 16$ , and  $2^3 = 8$  possible worlds, respectively. Clearly, this approach reduces the number of edges (marked in red in Figure 2(a)) that need to be sampled in each sampling iteration. More importantly, our experiments show that this approach of sampling component independently vastly decreases the variance of the total information flow, thus yielding a more precise estimation at the same number of samples.

Having defined syntax and semantics of the  $F$ -tree, the next section shows how to maintain the structure of a  $F$ -tree when additional edges are selected. It is important to note that we do not intend to insert all edges of a probabilistic graph  $\mathcal{G}$  into the  $F$ -tree. Rather, we only add the edges that are selected to compute the maximum flow  $\text{MaxFlow}(\mathcal{G}, Q, k)$  given a constrained budget of  $k$  edges. Thus, even in a case where all vertices are bi-connected, such as in the initial example in Figure 1(a), we note, supported by our experimental evaluation, that an optimal selection of edges prefers a spanning-tree-like topology, which synergizes well with our  $F$ -tree. The next section shows how to build the structure of the  $F$ -tree iteratively by adding edges to an initially empty graph.

The next subsection proposes an algorithm, to update a  $F$ -tree when a new edge is selected, starting at a trivial  $F$ -tree that contains only one component  $(\emptyset, Q)$ . Using this edge-insertion algorithm, we will show how to choose promising edges to be inserted to maximize the expected information flow. The selection of the edges of the  $F$ -tree will be shown in section 6.

## 5.4 Insertion of Edges into a F-tree

Following Definition 9 of a  $F$ -tree, each vertex  $v \in \mathcal{G}$  is assigned to either a single mono-connected component (noted by a flag  $v.\text{isMC}$  in the algorithm below), a single bi-connected component (noted by  $v.\text{isBC}$ ), or to no component, and thus disconnected from  $Q$ , noted by  $v.\text{isNew}$ . To insert a new edge  $(v_{src}, v_{dest})$ , our edge-insertion algorithm derived in this section differs between these cases as follows:

**Case I)**  $v_{src}.\text{isNew}$  and  $v_{dest}.\text{isNew}$ : We omit this case, as our edge selection algorithms presented in Section 6 always ensure a single connected component and initially the  $F$ -tree contains only vertex  $Q$ .

**Case II)**  $v_{src}.\text{isNew}$  exclusive-or  $v_{dest}.\text{isNew}$ : Due to considering undirected edges, we assume without loss of generality that  $v_{dest}.\text{isNew}$ . Thus  $v_{src}$  is already connected to  $F$ -tree.

**Case IIa)**  $v_{src}.\text{isMC}$ : In this case, a new dead end is added to the mono-connected structure  $MC_{src}$  which is guaranteed to remain mono-connected. We add  $v_{dest}$  to  $MC_{src}.V$ .

**Case IIb)**  $v_{src}.\text{isBC}$ : In this case, a new dead end is added to the bi-connected structure  $BC_{src}$ . This dead end becomes a new mono-connected component  $MC = (\{v_{dest}\}, v_{src})$ . Intuitively speaking, we know that vertex  $v_{dest}$  has no other choice but propagating its information to  $v_{src}$ . Thus,  $v_{src}$  becomes the articulation vertex of  $MC$ . The bi-connected component  $BC_{src}$  adds the new mono-connected component  $MC$  to its list of children.

**Case III)**  $v_{src}$  and  $v_{dest}$  belong to the same component, i.e.

$C_{src} = C_{dest}$

**Case IIIa)** This component is a bi-connected component  $BC$ : Adding a new edge between  $v_{src}$  and  $v_{dest}$  within component  $BC$  may change the reachability  $BC.P(v)$  of each vertex  $v \in BC.V$  to reach their articulation vertex  $BC.AV$ . Therefore,  $BC$  needs to be re-sampled to numerically estimate the reachability probability function  $P(v)$  for each  $v \in BC.V$ .

**Case IIIb)** This component is a mono-connected component  $MC$ : In this case, a new cycle is created within a mono-connected component, thus some vertices within  $MC$  may become bi-connected. We need to (i) identify the set of vertices affected by this cycle, (ii) split these vertices into a new bi-connected component, and (iii) handle the set of vertices that have been disconnected from  $MC$  by the new cycle. These three steps are performed by the *split-Tree*( $MC, v_{src}, v_{dest}$ ) function as follows: (i) We start by identifying the new cycle as follows: Compare the (unique) paths of  $v_{src}$  and  $v_{dest}$  to  $MC.AV$ , and find the first vertex  $v_{\wedge}$  that appears in both paths. Now we know that the new cycle is described by  $path(v_{\wedge}, v_{src}), path(v_{dest}, v_{\wedge})$  and the new edge between  $v_{src}$  and  $v_{dest}$ . (ii) All of these vertices are added to a bi-connected component  $BC = (path(v_{\wedge}, v_{src}) \cup path(v_{dest}, v_{\wedge}) \setminus v_{\wedge}, P(v), v_{\wedge})$  using  $v_{\wedge}$  as their articulation vertex. All vertices in  $MC$  having  $v_{\wedge}$  (except  $v_{\wedge}$  itself) on their path are removed from  $MC$ . The probability mass function  $P(v)$  is estimated by sampling the subgraph of vertices in  $BC.V$ . (iii) Finally, orphans of  $MC$  that have been split off from  $MC$  due to the creation of  $BC$  need to be collected into new mono-connected components. Such orphans having a vertex of the cycle  $BC$  on their path to  $MC.AV$  will be grouped by these vertices: For each  $v_i \in BC.V$ , let  $orphan_i$  denote the set of orphans separated by  $v_i$  (separated means  $v_i$  being the first vertex in  $BC.V$  on the path to  $MC.AV$ ). For each such group, we create a new mono-connected component  $MC_i = (orphan_i, v_i)$ . All these new mono-connected components with  $v_i \in BC.V$  become children of  $BC$ . If  $MC.V$  is now empty, thus all vertices of  $MC$  have been reassigned to other components, then  $MC$  is deleted and  $BC$  will be appended to the list of children of the component  $C$  where  $BC.AV = v_{\wedge} \in C.V$ . In case of  $MC.V$  being not empty, we are left over with a mono-connected component  $MC$  with  $v_{\wedge} \in MC.V$ . The new bi-connected component  $BC$  becomes a child of  $MC$ .

**Case IV)**  $v_{src}$  and  $v_{dest}$  belong to different components  $C_{src} \neq C_{dest}$ . Since the  $F$ -tree is a tree-structure itself, we can identify the lowest common ancestor  $C_{anc}$  of  $C_{src}$  and  $C_{dest}$ . The insertion of edge  $(v_{src}, v_{dest})$  has incurred a new cycle  $\bigcirc$  going from  $C_{anc}$  to  $C_{src}$ , then to  $C_{dest}$  via the new edge, and then back to  $C_{anc}$ . This cycle may cross mono-connected and bi-connected components, which all have to be adjusted to account for the new cycle. We need to identify all vertices involved to create a new cyclic, thus bi-connected, component for  $\bigcirc$ , and we need to identify which parts remain mono-connected. In the following cases, we adjust all components involved in  $\bigcirc$  iteratively. First, we initialize  $\bigcirc = (\emptyset, P, v_{anc})$ , where  $v_{anc}$  is the vertex within  $C_{anc}$  where the circle meets if  $C_{anc}$  is a mono-connected component, and  $C_{anc}.AV$  otherwise. Let  $C$  denote the component that is currently adjusted:

**Case IVa)**  $C = C_{anc}$ : In this case, the new circle may enter  $C_{anc}$  from two different articulation vertices. In this case, we apply *Case III*, treating these two vertices as  $v_{src}$  and  $v_{dest}$ , as these two vertices have become connected transitively via the big cycle  $\bigcirc$ .

**Case IVb)**  $C$  is a bi-connected component: In this case  $C$  becomes absorbed by the new cyclic component  $\bigcirc$ , thus  $\bigcirc.V = \bigcirc.V \cup C.V$ , and  $\bigcirc$  inherits all children from  $C$ . The rational is



that all vertices within  $C$  are able to access the new cycle.

**Case IVc)**  $C$  is a mono-connected component: In this case, one path in  $C$  from one vertex  $v$  to  $C.AV$  is now involved in a cycle. All vertices involved in  $path(v, C.AV)$  are added to  $\bigcirc.V$  and removed from  $C$ . The operation  $splitTree(C, v, C.AV)$  is called to create new mono-connected components that have been split off from  $C$  and become connected to  $\bigcirc$  via their individual articulation vertices.

In the following, we use the graph of Figure 2(a) and its corresponding  $F$ -tree representation of Figure 2(a) to insert additional edges and to illustrate the interesting cases of the insertion algorithm of Section 5.4.

## 5.5 Insertion Examples

In the following, we use the graph of Figure 2(a) and its corresponding  $F$ -tree (FT) representation of Figure 2(a) to insert additional edges and to illustrate the interesting cases of the insertion algorithm of Section 5.4.

We start by an example for **Case II** in Figure 3(a). Here, we insert a new edge  $a = (7, 17)$ , thus connecting a new vertex 17 to the FT. Since vertex 7 belongs to the bi-connected component  $BC$ , we apply **Case IIb**. A new mono-connected component  $G = (\{17\}, 7)$  is created, and added to the children of  $BC$ .

In Figure 3(b), we insert a new edge  $b = (6, 8)$  instead. In this case, the two connected vertices are already part of the FT, thus Case II does not apply. We find that both vertices belong to the same component  $C$ . Thus, **Case III** is used and more specifically, since component  $C$  is a bi-connected component  $BC$ , **Case IIIa** is applied. In this case, no components need to be changed, but the probability function  $BC.P(v)$  has to be re-approximated, as the probabilities of nodes 7, 8 and 9 will have increased probability of being connected to articulation vertex 6, due to the existence of new paths leading via edge  $b$ .

Next, in Figure 3(c), an edge is inserted between vertices 14 and 15. Both vertices belong to the mono-connected component  $E$ , thus **Case IIIb** is applied here. After insertion of edge  $c$ , the previously mono-connected component  $E = (\{13, 14, 15, 16\}, 9)$  now contains a cycle involving vertices 13, 14 and 15. (i) We identify this cycle by considering the previous paths from vertices 14 and 15 to their articulation vertex 9. These paths are  $(14, 13, 9)$  and  $(15, 13, 9)$ , respectively. The first common vertex on this path is 13, thus identifying the new cycle. (ii) We create a new bi-connected component  $G = (\{14, 15\}, 13)$ , containing all vertices of this cycle using the first common vertex 13 as articulation vertex. We further remove these vertices except the articulation vertex 13 from the mono-connected component  $E$ ; the probability function  $G.P(v)$  is initialized by sampling the reachability probabilities within  $G$ ; and component  $G$  is added to the list of children of  $E$ . (iii) Finally, orphans need to be collected. These are vertices in  $E$ , which have now become bi-connected to  $Q$ , because their (previously unique) path to their former articulation vertex 9 crosses a new cycle. We find that one vertex, vertex 16, had 15 as the first removed vertex on its path to 9. Thus, vertex 16 is moved from component  $E$  into a new mono-connected component  $H = (\{16\}, 15)$ , terminating this case. Summarizing, vertex 16 in component  $H$  now reports its information flow to vertex 15 in component  $G$ , for which the information flow to articulation vertex 13 in component  $G$  is approximated using Monte-Carlo sampling, this information is then propagated analytically to vertex 9 in component  $E$ , subsequently, the remaining flow that has been propagated all this way, is approximatively propagated to articulation vertex 6 in component  $C$ , which allows to analytically compute the flow to articulation vertex  $Q$ .

For the last case, **Case IV**, consider Figure 3(d), where a new edge  $d = (11, 15)$  connected two vertices belonging to two different components  $D$  and  $E$ . We start by identifying the cycle that has been created within the  $F$ -tree, involving components  $D$  and  $E$ , and meeting at the first common ancestor component  $C$ . For each of these components in the cycle  $(D, C, E)$ , one of the sub-cases of Case IV is used. For component  $C$ , we have that  $C = C_{anc}$  is the common ancestor component, thus triggering **Case IVa**. We find that both components  $D$  and  $E$  used vertex 9 as their articulation vertex  $v_{anc}$ . Thus, the only cycle incurred in component  $C$  is the (trivial) cycle  $(9)$  from vertex 9 to itself, which does not require any action. We initialize the new bi-connected component  $\bigcirc = (\emptyset, \perp, 9)$ , which initially holds no vertices, and has no probability mass function computed yet (the operator  $\perp$  can be read as null or not-defined) and uses  $v_{anc} = 9$  as articulation vertex. For component  $D$ , we apply **Case IVb**, as  $D$  is a bi-connected component, it becomes absorbed by a new bi-connected component  $\bigcirc$ , now having  $\bigcirc = (\{10, 11\}, \perp, 9)$ . For the mono-connected component  $E$  **Case IVc** is used. We identify the path within  $E$  that is now involved in a cycle, by using the path  $(15, 13, 9)$  between the involved vertex 15 to articulation vertex 9. All nodes on this path are added to  $\bigcirc$ , now having  $\bigcirc = (\{10, 11, 15, 13\}, \perp, 9)$ . Using the  $splitTree$  operation similar to Case III, we collect orphans into new mono-connected components, creating  $G = (\{14\}, 13)$  and  $H = (\{16\}, 15)$  as children of  $\bigcirc$ . Finally, Monte-Carlo sampling is used to approximate the probability mass function  $\bigcirc.P(v)$  for each  $v \in \bigcirc.V$ .

## 6. OPTIMAL EDGE SELECTION

The previous section presented the  $F$ -tree, a data structure to compute the expected information flow in a probabilistic graph. Based on this structure, heuristics to find a near-optimal set of  $k$  edges maximizing the information flow  $\text{MaxEFlow}(\mathcal{G}, Q, k)$  to a vertex  $Q$  (see Definition 4) are presented in this section. Therefore, we first present a Greedy-heuristic to iteratively add the locally most promising edges to the current result. Based on this Greedy approach, we present improvements, aiming at minimizing the processing cost while maximizing the expected information flow.

### 6.1 Greedy Algorithm

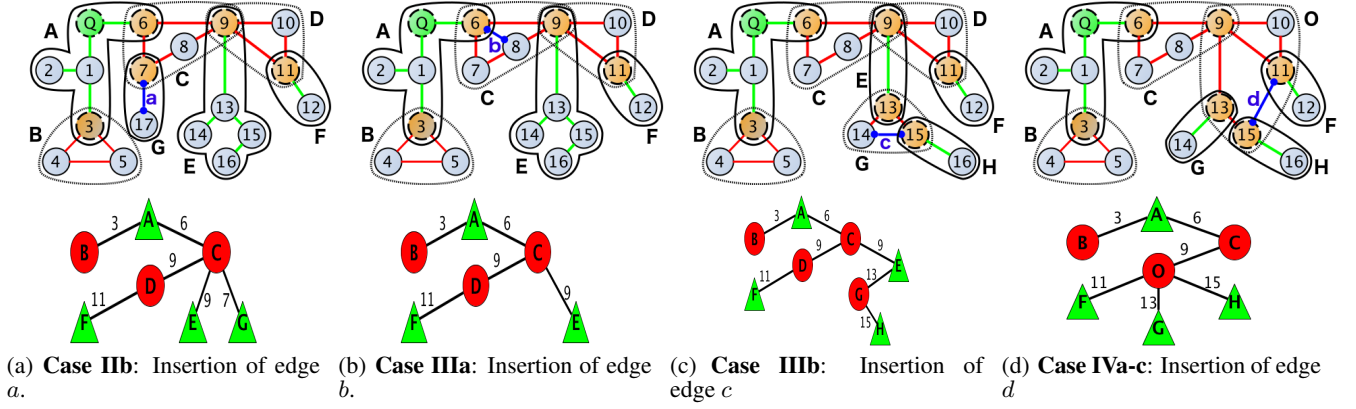
Aiming to select edges incrementally, the *Greedy* algorithm initially uses the probabilistic graph  $G_0 = (V, E_0 = \emptyset, P)$ , which contains no edges. In each iteration  $i$ , a set of candidate edges  $candList$  is maintained, which contains all edges that are connected to  $Q$  in the current graph  $G_i$ , but which are not already selected in  $E_i$ . Then, each iteration selects an edge  $e$  in addition which maximizes the information flow to  $Q$ , such that  $G_{i+1} = (V, E_i \cup e, P)$ , where

$$e = \underset{e \in candList}{\operatorname{argmax}} E(\text{flow}(Q, (V, E_i \cup e, P))). \quad (5)$$

For this purpose, each edge  $e \in candList$  is probed, by inserting it into the current  $F$ -tree using the insertion method presented in Section 5.3. Then, the gain in information flow incurred by this insertion is estimated by equation 1. After  $k$  iterations, the graph  $G_k = (V, E_k, P)$  is returned.

### 6.2 Component Memorization

We introduce an optimization reducing the number of bi-connected components for which their reachability probabilities have to be estimated using Monte-Carlo sampling, by exploiting stochastic independence between different components in the  $F$ -tree. During each Greedy-iteration, a whole set of edges  $candList$  is probed for



**Figure 3: Examples of edge insertions and  $F$ -tree update cases using the running example of Figure 2(a).**

insertion. Some of these insertions may yield new cycles in the  $F$ -tree, resulting from cases III and IV. Using component memorization, the algorithm memorizes, for each edge  $e$  in  $candList$ , the probability mass function of any bi-connected component  $BC$  that had to be sampled during the last probing of  $e$ . Should  $e$  again be inserted in a later iteration, the algorithm checks if the component has changed, in terms of vertices within that component or in terms of other edges that have been inserted into that component. If the component has remained unchanged, the sampling step is skipped, using the memorized estimated probability mass function instead.

### 6.3 Sampling Confidence Intervals

A Monte Carlo sampling is controlled by a parameter *samplesize* which corresponds to the number of samples taken to approximate the information flow of a bi-connected component to its articulation vertex. In each iteration, we can reduce the amount of samples by introducing confidence intervals for the information flow for each edge  $e \in candList$  that is probed. The idea is to prune the sampling of any probed edge  $e$  for which we can conclude that, at a sufficiently large level of significance  $\alpha$ , there must exist another edge  $e' \neq e$  in  $candList$  such that  $e'$  is guaranteed to have a higher information flow than  $e$ , based on the current number of samples only. To generate these confidence intervals, we recall that, following Equation 4 the expected information flow to  $Q$  is the sample-average of the sum of information flow of each individual vertex. For each vertex  $v$ , the random event of being connected to  $Q$  in a random possible world follows a binomial distribution, with an unknown success probability  $p$ . To estimate  $p$ , given a number  $S$  of samples and a number  $0 \leq s \leq S$  of 'successful' samples in which  $Q$  is reachable from  $v$ , we borrow techniques from statistics to obtain a two sided  $1 - \alpha$  confidence interval of the true probability  $p$ . A simple way of obtaining such confidence interval is by applying the Central Limit Theorem of Statistics to approximate a binomial distribution by a normal distribution.

**Definition 10** ( $\alpha$ -Significant Confidence Interval). *Let  $S$  be a set of possible graphs drawn from the probabilistic graph  $\mathcal{G}$ , and let  $\hat{p} := \frac{s}{S}$  be the fraction of possible graphs in  $S$  in which  $Q$  is reachable from  $v$ . With a likelihood of  $1 - \alpha$ , the true probability  $E(\uparrow(Q, v, \mathcal{G}))$  that  $Q$  is reachable from  $v$  in the probabilistic graph  $G$  is in the interval*

$$\hat{p} \pm z \cdot \sqrt{\hat{p}(1 - \hat{p})}, \quad (6)$$

where  $z$  is the  $100 \cdot (1 - 0.5 \cdot \alpha)$  percentile of the standard normal distribution. We denote the lower bound as  $E_{lb}(\uparrow(Q, v, \mathcal{G}))$  and the upper bound as  $E_{ub}(\uparrow(Q, v, \mathcal{G}))$ . We use  $\alpha = 0.05$ .

To obtain a lower bound of the expected information flow to  $Q$  in a graph  $G$ , we use the sum of lower bound flows of each vertex using Equation 4 to obtain

$$E_{lb}(\text{flow}(Q, \mathcal{G})) = \sum_{v \in V} E_{lb}(\uparrow(Q, v, \mathcal{G})) \cdot W(v)$$

as well as the upper bound

$$E_{ub}(\text{flow}(Q, \mathcal{G})) = \sum_{v \in V} E_{ub}(\uparrow(Q, v, \mathcal{G})) \cdot W(v)$$

Now, at any iteration  $i$  of the Greedy algorithm, for any candidate edge  $e' \in candList$  having an information flow lower bounded by  $lb := E_{lb}(\text{flow}(Q, \mathcal{G}_i \cup e'))$ , we prune any other candidate edge  $e' \in candList$  having an upper bound  $ub := E_{ub}(\text{flow}(Q, \mathcal{G}_i \cup e'))$  if  $lb > ub$ . The rational of this pruning is that, with a confidence of  $1 - \alpha$ , we can guarantee that inserting  $e'$  yields less information gain than inserting  $e$ . To ensure that the Central Limit Theorem is applicable, we only apply this pruning step if at least 30 sample worlds have been drawn for both probabilistic graphs.

### 6.4 Delayed Sampling

For the last heuristic, we reduce the number of Monte-Carlo samplings that need to be performed in each iteration of the Greedy Algorithm in Section 6.1. In a nutshell, the idea is that an edge, which yields a much lower information gain than the chosen edge, is unlikely to become the edge having the highest information gain in the next iteration. For this purpose, we introduce a delayed sampling heuristic. In any iteration  $i$  of the Greedy Algorithm, let  $e$  denote the best selected edge, as defined in Equation 5. For any other edge  $e' \in candList$ , we define its potential  $pot(e') := \frac{E(\text{flow}(Q, (V, E_i \cup e', P)))}{E(\text{flow}(Q, (V, E_i \cup e, P)))}$ , as the fraction of information gained by adding edge  $e'$  compared to the best edge  $e$  which has been selected in an iteration. Furthermore, we define the cost  $cost(e')$  as the number of edges that need to be sampled to estimate the information gain incurred by adding edge  $e'$ . If the insertion of  $e'$  does not incur any new cycles, then  $cost(e')$  is zero. Now, after iteration  $i$  where edge  $e'$  has been probed but not selected, we define a sampling delay

$$d(e') = \lfloor \log_c \frac{cost(e')}{pot(e')} \rfloor,$$

which implies that  $e'$  will not be considered as a candidate in the next  $d$  iterations of the Greedy algorithm of Section 6.1. This definition of delay, makes the (false) assumption that the information



gain of an edge can only increase by a factor of  $c > 1$  in each iteration, where the parameter  $c$  is used to control the penalty of having high sampling cost and having low information gain. As an example, assume an edge  $e'$  having an information gain of only 1% of the selected best edge  $e$ , and requiring to sample a new bi-connected component involving 10 edges upon probing. Also, we assume that the information gain per iteration (and thus by insertion of other edges in the graph), may only increase by a factor of at most  $c = 2$ . We get  $d(e') = \lfloor \log_{\frac{10}{0.01}} \rfloor = \lfloor \log_2 1000 \rfloor = 9$ . Thus, using delayed sampling and having  $c = 2$ , edge  $e'$  would not be considered in the next nine iterations of the edge selection algorithm. It must be noted that this delayed sampling strategy is a heuristic only, and that no correct upper-bound  $c$  for the change in information gain can be given. Consequently, the delayed sampling heuristic may cause the edge having the highest information gain not to be selected, as it might still be suspended. Our experiments show that even for low values of  $c$  (i.e., close to 1), where edges are suspended for a large number of iterations, the loss in information gain is fairly low.

## 7. EVALUATION

This section evaluates efficiency and effectiveness of our proposed solutions to compute a near-optimal subgraph of an uncertain graph to maximize the information flow to a source node  $Q$ , given a constrained number of edges, according to Definition 4. As motivated in the introductory Section 1, two main application fields of information propagation on uncertain graphs are: i) information/data propagation in spatial networks, such as wireless networks or a road networks, and ii) information/belief propagation in social networks. These two types of uncertain graphs have extremely different characteristics, which require separate evaluation. A spatial network follows a locality assumption, constraining the set of pairwise reachable nodes to a spatial distance. Thus, the average shortest path between a pair of two randomly selected nodes can be very large, depending on the spatial distance. In contrast, a social network has no locality assumption, thus allowing to move through the network with very few hops. As a result, without any locality assumption, the set of nodes reachable in  $k$ -hops from a query node may grow exponentially large in the number of hops. In networks following a locality assumption, this number grows polynomial, usually quadratic (in sensor and road networks on the plane) in the range  $k$ , as the area covered by a circle is quadratic to its radius. Our experiments have shown, that the locality assumption, which clearly exists in some applications but not in others, has tremendous impact on the performance of our algorithms, including the baseline. Consequently, we evaluate both cases separately. Beside these two cases we also evaluate the following parameters, with default values specified as follows: size of the Graph  $|V| = 10,000$ , average vertex degree  $d = 2$ , and the budget of edges  $k = 100$ .

All experiments were evaluated on a system with Windows 10, 64Bit, 16.0 GB RAM with the processor unit Intel(R) Xeon(R) CPU E3-1220, 3.10 Ghz. All algorithms were implemented in Java (version 1.8.0\_91).

### 7.1 Dataset Descriptions

This section describes our employed uncertain graph datasets. For both the case of locality assumption and no-locality assumption, we use synthetic and real datasets.

**Synthetic Datasets: No locality assumption.** Our first model denoted as *Erdős* is based on the idea of the *Erdős-Rényi model* [8], distributing edges independently and uniformly between nodes. Probabilities of edges are chosen uniformly in  $[0, 1]$  and weights of

nodes are integers selected uniformly from  $[0, 10]$ . It is known that this model is not able to capture real human social networks [21], due to the lack of modeling of long tail distributions produced by “social animals”. Thus, we use this data generation only in our first set of experiments, using real social network data later.

**Synthetic Datasets: Locality assumption.** We use two synthetic data generating scheme to generate spatial networks. For the first data generating scheme, denoted by *partitioned*, each vertex has the same degree  $d$ . The dataset is partitioned into  $n = 2 \cdot \frac{|V|}{d}$  partitions  $P_0, \dots, P_{n-1}$  of size  $d$ . Each vertex in partition  $P_i$  is connected to all and only vertices in the previous and next partition  $P_{(i-1) \bmod n}$  and  $P_{(i+1) \bmod n}$ . This data generation allows to control the diameter of a resulting network, which is guaranteed to be equal to  $n - 1$ .

For a more realistic synthetic data set denoted as *WSN*, we simulate a wireless sensor network. Here, vertices have two spatial coordinates selected uniformly in  $[0, 1]$ . Using a global parameter  $\epsilon$ , any vertex  $v$  is connected to all and only vertices located in the  $\epsilon$  distance of  $v$  using Euclidean distance.

For both settings the probabilities of edges are chosen uniformly in  $[0, 1]$ .

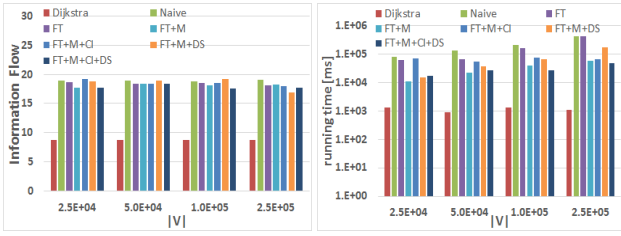
**Real Datasets: No locality assumption.** We use the *social circles of Facebook dataset* published in [24]. This dataset is a snapshot of the social network of Facebook - containing a subgroup of 535 users which form a social circle, i.e., a highly connected subgraph, having  $10k$  edges. These users have excessive number of ‘friends’. Yet, it has been discussed in [37] that the number of real friends that influence, affect and interact with an individual is limited. According to this result, and due to the lack of better knowledge which people of this social network are real friends, we applied higher edge probabilities uniformly selected in  $[0.5; 1.0]$  to 10 random adjacent nodes of each user. Due to symmetry, an average user has 20 such high probabilities ‘close friends’. All other edges are assigned edge probabilities uniformly selected in  $[0; 0.5]$ .

For our experiments on *collaboration network data*, we used the scientific collaborations between authors who submitted papers to the General Relativity and Quantum Cosmology category. The structure of this dataset is that if an author  $v_i$  co-authored a paper with author  $v_j$ , where  $i \neq j$ , the graph contains a undirected edge  $e$  from  $v_i$  to  $v_j$ . If a paper is co-authored by  $k$  authors this generates a completely connected (sub)graph on  $k$  nodes. This dataset has been published in [23]. The data covers papers in the period from January 1993 to April 2003 (124 months). Probabilities on edges are uniformly distributed in  $[0; 1]$ . The graph consists of  $|V| = 5242$  vertices and  $|E| = 14496$  edges.

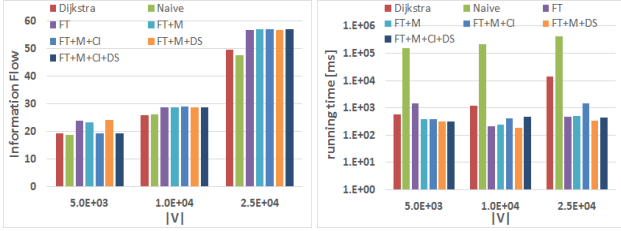
Finally, we evaluated our methods also on the Youtube social network, first published in [26]. In this network, edges represent friendship of the users with each other. The graph consists of  $|V| = 1134890$  vertices and  $|E| = 2987624$  edges. Again, the probabilities on edges are uniformly distributed in  $[0; 1]$ .

**Real Datasets: Locality assumption.** For our experiments on spatial networks we used the road network of San Joaquin County<sup>1</sup>, having  $|V| = 18263$  vertices and  $|E| = 23874$  edges. The vertices of the graph are road intersections and edges correspond to connections between them. In order to simulate real sensor nodes located at road intersections, we have connected vertices that are spatially distant from each other have a lower chance to successfully communicate. Specifically, for two vertices having a distance of  $a$  in meters, we set the communication probability to  $e^{-0.001a}$ . Thus, a  $10m$ ,  $100m$  and  $1km$  distance will yield a probability of  $e^{-0.01} = 99\%$ ,  $e^{-0.1} = 90\%$ , and  $e^{-1} = 36\%$ , respectively.

<sup>1</sup><https://www.cs.utah.edu/lifeifei/SpatialDataset.htm>



(a) Changing Graph Size with locality assumption



(b) Changing Graph Size without locality assumption

Figure 4: Experiments with changing graph size

## 7.2 Evaluated Algorithms

The algorithms that we evaluate in this section are denoted and described as follows:

**Naive** As proposed in [22, 7] the first competitor *Naive* does not utilize the strategy of component decomposition of Section 5 and utilizes a pure sampling approach to estimate reachability probabilities. To select edges, the naive approach chooses the locally best edge as shown in Section 6 but does not use the *F-tree* representation presented in Section 5.3. We use a constant Monte-Carlo sampling size of 5000 samples.

**Dijkstra** Shortest-path spanning trees [35] are used to interconnect a wireless sensor network to a sink node. To obtain a maximum probability spanning tree, we proceed as follows: the cost  $w(e)$  of each edge  $e \in E$  is set to  $w(e) = -\log(P(e))$ . Running the traditional Dijkstra algorithm on the transformed graph starting at node  $Q$  yields, in each iteration, a spanning tree which maximizes the connectivity probability between  $Q$  and any node connected to  $Q$  [33]. Since, in each iteration, the resulting graph has a tree structure, this approach can fully exploit the concept of Section 5, requiring no sampling step at all.

**FT** employs the *F-tree* proposed in Section 5.3 to estimate reachability probabilities. To sample bi-connected components, we draw 5000 samples for a fair comparison to *Naive*. All following FT-Algorithms build on top of *FT*.

**FT+M** additionally maintains for each candidate edge  $e$  the *pdf* of the corresponding bi-connected component from the last iteration (cf. Section 6.2).

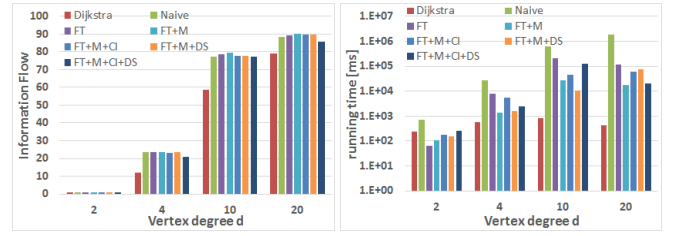
**FT+M+CI** ensures that probing of an edge is stopped whenever another edge has a higher information flow with a certain degree of confidence as explained in Section 6.3.

**FT+M+DS** tries to minimize the candidate edges in an iteration by leaving out edges that had a small information gain/cost-ratio in the last iteration (cf. Section 6.4). Per default, we set the penalization parameter to  $c = 2$ .

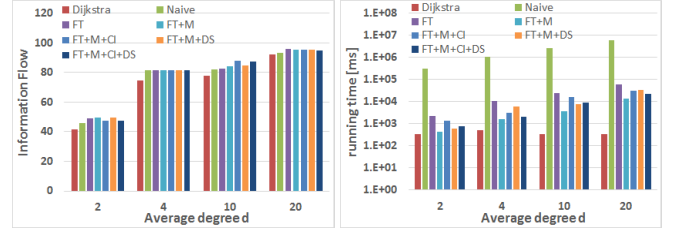
**FT+M+CI+DS** Combines all of the above concepts.

## 7.3 Experiments on Synthetic Data

In this section, we perform experiments on randomly generated uncertain graphs. We generate graphs having no-locality-assumption using *Erdős* graphs and having locality assumption using the *partitioned* generation. Both generation approaches are described in



(a) Changing Graph Density with locality assumption

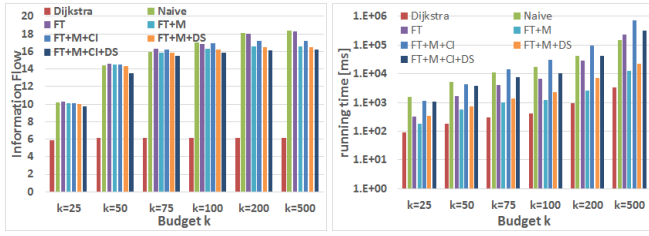


(b) Changing Graph Density without locality assumption

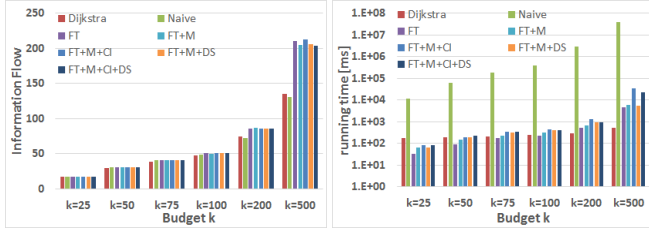
Figure 5: Experiments with changing graph density

Section 7.1. This data generation allows us to scale the topology of the uncertain graph  $\mathcal{G}$  in terms of size and density. Per default, we use  $|V| = 1000$ ,  $|E| = 4 \cdot |V|$  and  $k = 100$ .

**Graph Size.** We first scale the size  $|V|$  of the synthetic graphs. Figure 4(a) shows the information flow (left-hand-side) and run-time (right-hand-side) for our synthetic data set following the locality assumption. First, we note that the Dijkstra-based shortest-path spanning tree yields an extremely low information flow, far inferior to all other approaches. The reason is that such a tree structure allows no room for failure of edges: whenever any edge fails, the whole subtree become disconnected from  $Q$ . We further note that all other algorithms, including the naive one, are oblivious to the size of the network, in terms of information flow and run-time. The reason is that, due to the locality assumption, only a local neighborhood of vertices and edges is relevant, regardless of the global size of the graph. Additionally, we see that confidence interval heuristics (CI) yields an significant run-time performance gain, but at the cost of a severe loss of information flow. Figure 4(b), shows the performance in terms of information gain and run-time for the *Erdős* graphs having no locality assumption. We first observe that *Dijkstra* and *Naive* yield a significantly lower information flow than our proposed approaches. For *Dijkstra*, this result is again contributed to the constraint of constructing a spanning tree, and thus not allowing any edges to connect the flow between branches. For the *Naive* approach, the loss in information flow requires a closer look. This approach samples the whole graph 5000 times, to estimate the information flow. In contrast, our *F-tree* approach samples each individual bi-connected component 5000 times. Why is the later approach more accurate? A first, informal, explanation is that, for a constant sampling size, the information flow of a small component can be estimated more accurately than for a large component. Intuitively, sampling two independent components  $n$  times each, yields a total of  $n^2$  combinations of samples of their joint distribution. More formally, this effect is contributed to the fact that the variance of the sum of two random variables increases as their correlation increases, since  $Var(\sum_{i=1}^n X_i) = \sum_{i=1}^n Var(X_i) + 2 \sum_{1 \leq i < j \leq n} Cov(X_i, X_j)$  [27]. Furthermore, the naive approach also incurs an approximation error for mono-connected components, for which all *F-tree* (FT) approaches compute the exact flow analytically. We further see that that the *Naive* approach, which has to sample the whole

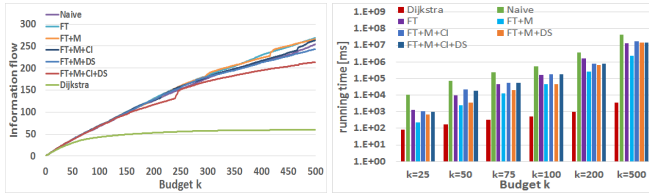


(a) Changing budget  $k$  with locality assumption

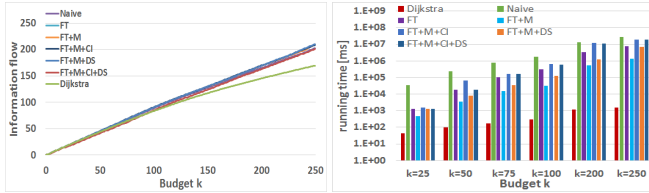


(b) Changing budget  $k$  without locality assumption

Figure 6: Experiments with changing Budget



(a) WSN  $\epsilon = 0.05$



(b) WSN  $\epsilon = 0.07$

Figure 7: Experiments in synthetic Wireless Sensor Networks

graph, is by far the most inefficient. On the other end of the scope, the *Dijkstra* approach, which is able to avoid sampling entirely by guaranteeing a single mono-connected component, is the fastest in all experiments, but at the cost of information flow. In the case of a graph having no locality assumption, this loss of information flow is much less dramatic than in cases satisfying the locality assumption. We also see that in Figure 4(b) all algorithms increase their run-time and information flow as the graph increases. This is due to the fact that more nodes in such a graph give more and better options to choose from, thus yielding more candidates to be explored.

**Graph Density.** In this experiment, we scale the average degree of vertices. In the case of graphs following the locality assumption, the gain in information flow of all proposed solutions compared to *Dijkstra* is quite significant as shown in Figure 5(a), particularly when the degree of vertices is low. This is the case in road networks and most sensor and adhoc networks. The reason is that, in such case, spanning trees gain quickly in height as edges are added, thus incurring low-probability paths that require circular components to connect branches to support the information flow. For larger vertex degrees, the optimal solutions become more star-like, thus becoming more tree-like. In the case of  $d = 2$  we see a degeneration of the *partitioned* data generation to a ring. Here, each partition con-

tains exactly one node, and edges are added only to neighbors. In this case, the chance of any node submitting its information to  $Q$  drops exponentially in the distance on the ring. The results shown in Figure 5(b) indicate that the *Dijkstra* approach is able to find higher quality results in graphs without locality assumption. This is contributed to the fact that in graphs with no locality assumption the optimal result will be almost tree-like, having only a few inter-branch-edges. In this case, the optimal subtree, which is returned by *Dijkstra* is close to the optimal subgraph in terms of information gain. In both scenarios *FT+M* yields a good trade-off between runtime and accuracy while the *DS* and *CI* pruning cost time but do not yield better results.

**Scaling of parameter  $k$ .** In the next experiments, shown in Figure 6, we show how the budget  $k$  of edges affects the performance of the proposed algorithms. In the case of a network following the locality assumption, we observe in Figure 6(a) that the overall information gain per additional edge slowly decreases. This is clear, since, in average, the hop distance to  $Q$  increases as more edges are added, increasing the possible links of failure, thus decreasing the likelihood of propagating a nodes information to  $Q$ . We observe that the effectiveness of *Dijkstra* in the locality setting quickly deteriorates, since the constraint of returning a tree structure leads to paths between  $Q$  and other connected nodes that become increasingly unlikely, making the *Dijkstra* approach unable to handle such settings. The *Naive* approach yields high information gain, but suffers from high run-time. In this setting, we observe that the memorization heuristics  $M$  applied to our  $F$ -tree perform extremely well: At the cost of very little information gain, the reduction in run-time reaches an order of magnitude. Also, we observe that at a high budget  $k$ , the heuristics of using confidence intervals  $CI$  has a detrimental effect on the run-time. Thus, the overhead for estimating and updating lower- and upper-bounds of the confidence intervals often out-weights the gain obtained from stopping Monte-Carlo sampling early. Overall, our  $F$ -tree approach using only the memorization heuristic performs best in this setting using locality assumption.

In contrast, using data following no locality assumption in Figure 6(b), we see that both *Dijkstra* and *Naive* yield an extremely low information gain for a large budget  $k$ . For *Dijkstra*, the reason is that for large values of  $k$ , the depths of the spanning tree, which is lower bounded by  $\log_d k$ , incurs longer paths without any backup route in the case of a single failure. For the *Naive*-approach, the low information gain is contributed in the high variance of sampling the information flow of the whole graph for each edge selection. Further, we see that the *Naive* approach further suffers from an extreme run-time, requiring to re-sample the whole graph in each iteration, thus incurring a run-time super-linear in  $k$ : without locality assumption, the number of candidates, which have to be probed in each iteration increases linear in the number of added edges, yielding at least total quadratic run-time cost. In contrast, the  $F$ -tree and all its heuristic give a consistently high information gain in this case of having no locality assumption, while having a consistently low run-time.

Overall, we summarize that our  $F$ -tree-based approaches maintain high information flow for large values of  $k$ . At the same time, the extremely high run-time of *Naive* is avoided on data having no locality assumption. We further see that the additional heuristics presented in Section 6 can be used to adjust this trade-off between effectiveness and efficiency.

**Synthetic Wireless Sensor Networks (WSN).** In this experiments, we simulate real world wireless sensor networks. We embed a number of vertices - here  $|V| = 1000$  - according to a uniform distribution in a spatial space  $[0; 1] \times [0; 1]$ . For each vertex, we observe adjacent vertices being in its proximity which is reg-

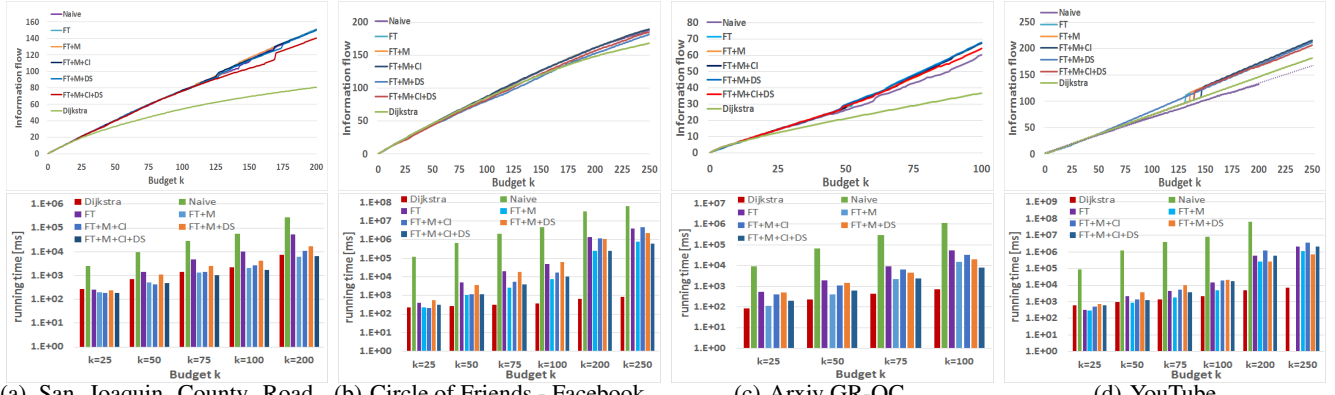


Figure 8: Experiments on Real World Datasets

ulated by an additional parameter  $\epsilon$ . Figure 7 shows the results. We observe nearly the same behavior as in the settings including a locality assumptions using the *partitioned* data generator. By increasing the parameter  $\epsilon$ , hence, simulating very dense graphs, the gap between *Dijkstra* and the *F-tree* approaches is reduced. For these datasets we can also observe the benefit of *FT+M+CI+DS* which still identifies the optimal information gain whilst reducing the running time, as the number of candidates are reduced, respectively, we can prune candidates in earlier stages of each iteration.

## 7.4 Experiments on Real World Data

**San Joaquin County Road Network.** As road networks are of very sparse nature, and follow a strong locality assumption, our approaches outperforms *Dijkstra* significantly as  $k$  is scaled to  $k = 200$ . Thus, *Dijkstra* is highly undesirable as budget is wasted without proper return in information flow. Among all other approaches, *FT+M+CI+DS* is the best in terms of the running time. As it uses all heuristics to speed up computation. This comes at the cost of a slightly gap in the information flow compared to the other methods. The algorithm *FT+M+CI* is the best when  $k < 100$  regarding running time. As the list of candidates increases - and therefore more confidence intervals have to be considered - the running time suffers. We note that using our default parameters, the confidence interval heuristics does yield an increase in run-time, as the overhead to maintain the intervals does not warrant the pruning power. This can be solved by using a lower significance level  $\alpha$ , but at a further loss of information flow.

**Circles of Friends - Facebook.** The *social circles of friends* dataset is an extremely dense network with no locality assumption, where most pairs of nodes are connected. Figure 8(b) shows that *Dijkstra* maintains high information flow much longer, as most of the  $k$  selected edges will be directly connected to  $Q$ , or connected via a single direct friend. At around  $k = 160$  the information gain of all “close” friends and “close friends of close friends” have been exhausted, thus *Dijkstra* is now enforced to take edges with a low information gain for spanning its MST which results again in a sublinear behavior.

**Arxiv GR-QC.** Figure 8(c) shows our results on the collaboration network dataset, a sparse network which follows no locality assumption. Again, we observe a rapid loss of potential information flow for *Dijkstra* as  $k$  increases. In this experiment, the approach *FT+M+CI+DS* outperforms all other approaches in terms of running time. Again, the running time of *FT+M+CI* suffers as more candidates have to be regarded. All other heuristics buy run-time at the cost of information flow, as we expect.

**YouTube.** Likewise to the discussion before, we observe similar behavior of all approaches even on bigger graphs like the *YouTube*

*social network*, which refers to a sparse setting with no locality assumption. Figure 8(d) shows the results. As in the other settings, we can observe the beginning divergence between our proposed methods and *Dijkstra* as the budget of  $k$  edges increases. Due to the high run-time of the *Naive* approach, we omitted the results for large  $k$ . As  $k$  increases the heuristic of a delayed sampling *FT+M+DS* outperforms the other approaches.

Finally, we also evaluated the penalization parameter  $c$  of the delayed sampling heuristics and summarize the results. In all our evaluated settings, ranging from  $1.01 \geq c \geq 16$ , the run-time consistently decreases as  $c$  is decreased, yielding a factor of 2 to 10 speed-up for  $c = 1.2$ , depending on the dataset, and a multi-orders of magnitude speed-up for  $c = 1.01$ . Yet, for  $c < 1.2$  we start to observe a significant loss of information flow. For the extreme case of  $c = 1.01$ , the information flow became worse than *Dijkstra*, as edges become suspended unreasonable long, choosing edges nearly arbitrarily. For the default setting of  $c = 2$  used all previous evaluations, the delayed sampling heuristics showed insignificant loss of information, but yielding

## 8. CONCLUSIONS

In this paper we discussed solutions for the problem of maximizing information flow in an uncertain graph given a fixed budget of  $k$  communication edges. We identified two NP-hard subproblems that needed heuristical solutions: (i) Computing the expected information flow of a given subgraph, and (ii) selecting the optimal  $k$ -set of edges. For problem (i) we developed an advanced sampling strategy that only performs an expensive (and approximate) sampling step for parts of the graph for which we can not obtain an efficient (and exact) analytic solution. For problem (ii) we propose our *F-tree* representation of a graph  $G$ , which keeps track of *bi-connected components* - for which sampling is required to estimate the information flow - and *mono-connected components* - for which the information flow can be computed analytically. On the basis of the *F-tree* representation, we introduced further approaches and heuristics to handle the trade-off between effectiveness and efficiency. Our evaluation shows that these enhanced algorithms are able to find high quality solutions (i.e.,  $k$ -sets of edges having a high information flow) in efficient time, especially in graphs following a locality assumption, such a road networks and wireless sensor networks.

## 9. REFERENCES

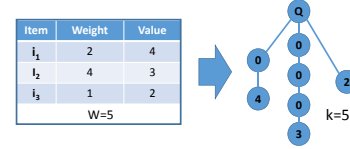
- [1] E. Adar and C. Ré. Managing uncertainty in social networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.
- [2] K. Aggarwal, K. Misra, and J. Gupta. Reliability evaluation a comparative study of different techniques. *Microelectronics Reliability*, 14(1):49–56, 1975.



- [3] T. B. Brecht and C. J. Colbourn. Lower bounds on two-terminal network reliability. *Discrete Applied Mathematics*, 21(3):185–198, 1988.
- [4] D. Bulka and J. B. Dugan. Network st reliability bounds using a 2-dimensional reliability polynomial. *Reliability, IEEE Transactions on*, 43(1):39–45, 1994.
- [5] C. J. Colbourn and C. Colbourn. *The combinatorics of network reliability*, volume 200. Oxford University Press New York, 1987.
- [6] P. Domingos and M. Richardson. Mining the network value of customers. In *SIGKDD*, pages 57–66, 2001.
- [7] T. Emrich, H.-P. Kriegel, J. Niedermayer, M. Renz, A. Suhartha, and A. Züfle. Exploration of monte-carlo based probabilistic query processing in uncertain graphs. In *CIKM*, pages 2728–2730, 2012.
- [8] P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication of the mathematical institute of the hungarian academy of sciences*, pages 17–61, 1960.
- [9] J. Galtier, A. Laugier, and P. Pons. Algorithms to evaluate the reliability of a network. In *DRCN*, pages 8–pp, 2005.
- [10] J. Ghosh, H. Q. Ngo, S. Yoon, and C. Qiao. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM*, pages 1721–1729, 2007.
- [11] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *WWW*, pages 403–412, 2004.
- [12] G. Hardy, C. Lucet, and N. Limnios. K-terminal network reliability measures with binary decision diagrams. *Reliability, IEEE Transactions on*, 56(3):506–515, 2007.
- [13] P. Hintsanen. The most reliable subgraph problem. In *PKDD*.
- [14] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.
- [15] R. Jin, L. Liu, and C. C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *SIGKDD*, pages 992–1000, 2011.
- [16] M. Kasari, H. Toivonen, and P. Hintsanen. Fast discovery of reliable k-terminal subgraphs. In M. J. Zaki, J. X. Yu, B. Ravindran, and V. Pudi, editors, *PAKDD*, volume 6119, pages 168–177, 2010.
- [17] H. Kellerer, U. Pferschy, and D. Pisinger. *Introduction to NP-Completeness of knapsack problems*. Springer, 2004.
- [18] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, 2003.
- [19] A. Khan, F. Bonchi, A. Gionis, and F. Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014.
- [20] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *Knowledge and Data Engineering, IEEE Transactions on*, 25(2):325–336, 2013.
- [21] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *ECML-PKDD*, pages 133–145. Springer, 2005.
- [22] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *SIGKDD’06*, pages 631–636, 2006.
- [23] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*.
- [24] J. Leskovec and J. J. McAuley. Learning to discover social circles in ego networks. In *NIPS*, pages 539–547, 2012.
- [25] J. Li, Z. Zou, and H. Gao. Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *The VLDB Journal*, 21(6):753–777, 2012.
- [26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC’07)*, San Diego, CA, October 2007.
- [27] W. C. Navidi. *Statistics for engineers and scientists*. McGraw-Hill New York, 2006.
- [28] O. Papapetrou, E. Ioannou, and D. Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *EDBT*, pages 355–366, 2011.
- [29] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. *PVLDB*, 3(1):997–1008, 2010.
- [30] J. S. Provan and M. O. Ball. Computing network reliability in time polynomial in the number of cuts. *Operations Research*, 32(3):516–526, 1984.
- [31] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *SIGKDD*, pages 61–70, 2002.
- [32] G. Rubino. Network reliability evaluation. *State-of-the art in performance modeling and simulation*, pages 275–302, 1998.
- [33] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *DILS*, pages 35–49, 2006.
- [34] A. R. Sharafat and O. R. Ma’rouzi. All-terminal network reliability using recursive truncation algorithm. *Reliability, IEEE Transactions on*, 58(2):338–347, 2009.
- [35] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE personal communications*, 7(5):16–27, 2000.

- [36] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.
- [37] B. Wellman, A. Q. Haase, J. Witte, and K. Hampton. Does the internet increase, decrease, or supplement social capital? social networks, participation, and community commitment, 2001.
- [38] J. Westbrook and R. E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, 1992.
- [39] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient keyword search on uncertain graph data. *TKDE*, 25(12):2767–2779, 2013.
- [40] Z. Zou, H. Gao, and J. Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *SIGKDD*, pages 633–642, 2010.
- [41] Z. Zou, J. Li, H. Gao, and S. Zhang. Finding top-k maximal cliques in an uncertain graph. In *ICDE*, pages 649–652, 2010.

## 10. APPENDIX



**Figure 9: Example of the Knapsack Reduction of Theorem 1**  
**Theorem 3.** Even if the expected information flow( $Q, \mathcal{G}$ ) to a vertex  $Q$  can be computed in  $O(1)$  for any probabilistic graph  $\mathcal{G}$ , the problem of finding  $\text{MaxFlow}(\mathcal{G}, Q, k)$  is NP-hard.

*Proof.* In this proof, we will show that a special case of computing  $\text{MaxFlow}(\mathcal{G}, Q, k)$  is NP-complete, thus implying that our general problem is NP-hard. We reduce the 0-1 knapsack problem to the problem of computing  $\text{MaxFlow}(\mathcal{G}, Q, k)$ . Thus, assume a 0-1 knapsack problem: Given a capacity integer  $W$  and given a set  $\{i_1, \dots, i_n\}$  of  $n$  items each having an integer weight  $w_i$  and an integer value  $v_i$ . The 0-1 knapsack problem is to find the optimal vector  $x = (x_1, \dots, x_n) \in \{0, 1\}^n$  such that  $\sum_{i=1}^n v_i \cdot x_i$ , subject to  $\sum_{i=1}^n w_i \cdot x_i \leq W$ . This problem is known to be NP-complete [17]. We reduce this problem to the problem of computing  $\text{MaxFlow}(\mathcal{G}, Q, k)$  as follows. Let  $\mathcal{G} = (V, E, G)$  be a probabilistic graph such that  $Q$  is connected to  $n$  nodes  $\{V_1, \dots, V_n\}$  (one node for each item of the knapsack problem). Each node  $V_i$  is connected to a chain of  $w_i - 1$  nodes  $\{V_i^1, \dots, V_i^{w_i-1}\}$ . All edges have a probability of one, i.e.,  $P(v \in V) = 1$ . The information of a node is set to  $w_i$  if it is the (only) leaf node  $V_i^{w_i-1}$  of the branch of  $\mathcal{G}$  connected to  $V_i$  and zero otherwise. Finally, set  $k = W$ . Then, the solution of the 0-1 knapsack problem can be derived from the constructed  $\text{MaxFlow}(\mathcal{G}, Q, k)$  problem by selected all items  $n_i$  such that the corresponding node  $V_i^{w_i-1}$  is connected to  $Q$ . Thus, if we can solve the  $\text{MaxFlow}(\mathcal{G}, Q, k)$  problem in polynomial time, then we can solve the 0-1 knapsack problem in polynomial time: A contradiction assuming  $P \neq NP$ .  $\square$

**Lemma 4.** In a bi-connected graph  $\mathcal{G}$  of size  $|V| \geq 3$ , all pairs of vertices are bi-connected following Definition 7.

*Proof.* By contradiction, let  $A, B$  be two nodes in  $\mathcal{G}$  that are mono-connected. Let  $\text{path}(A, B)$  be the only path between them. Case 1:  $\text{path}(A, B) = (A, B)$  contains no other vertices: Since  $\mathcal{G}$  is bi-connected, removal of vertex  $A$  yields a graph where  $B$  and  $C$  are connected by some path  $\text{path}(C, B)$ . At the same time, removal of vertex  $B$  yields a graph where  $A$  and  $C$  are connected by some path  $\text{path}(A, C)$ . Thus, the concatenation of these paths yields an alternative path between  $A$  and  $B$ , contradicting the assumption that  $(A, B)$  are mono-connected by path  $(A, B)$ . Case 2:  $\text{path}(A, B) = (A, C_1, \dots, C_n, B)$  contains other vertices. Let  $C_1$  be such a vertex. Since  $\mathcal{G}$  is bi-connected, removal of vertex  $C_1$  yields a graph where  $A$  and  $B$  are still connected, contradicting the assumption that  $A$  and  $B$  are mono-connected by path  $(A, B)$  only.  $\square$